Waters
THE SCIENCE OF WHAT'S POSSIBLE.™

Large and Sparse – Mass Spectrometry Data Processing in the GPU

Jose de Corral

2012 GPU Technology Conference

Agenda

- Overview of LC/IMS/MS
- 3D Data Processing
- 4D Data Processing
- Results and Conclusions

Overview of LC/IMS/MS

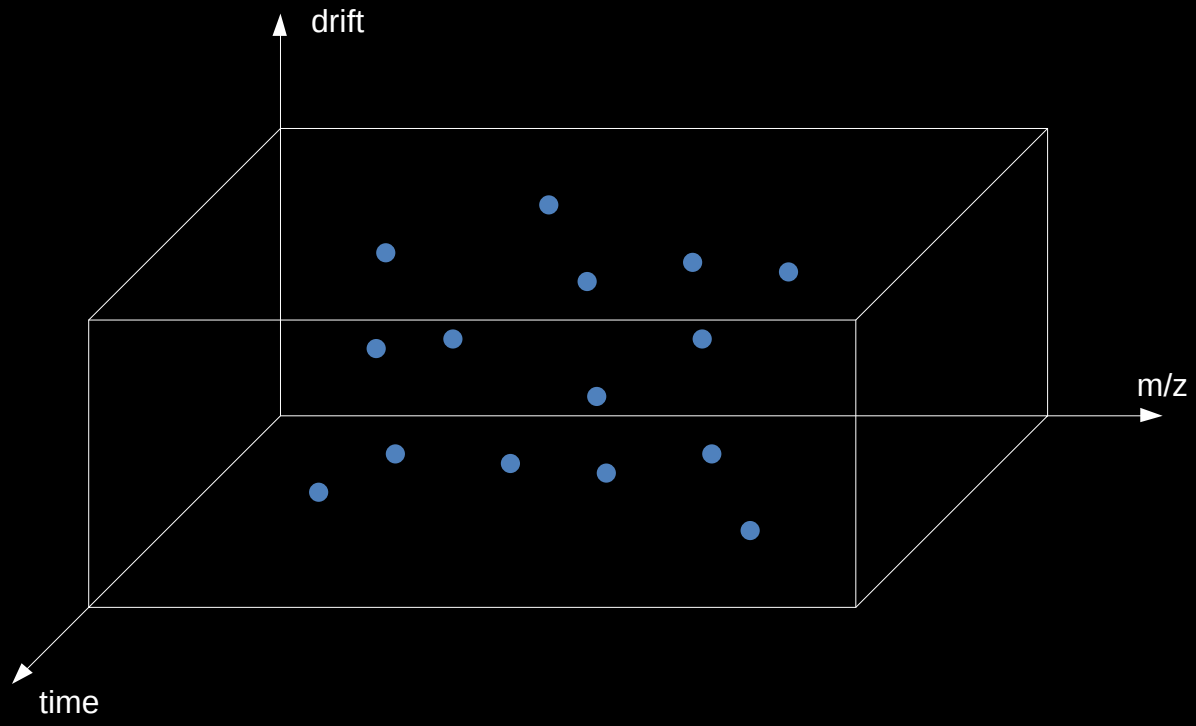
Overview of LC/IMS/MS

Waters
THE SCIENCE OF WHAT'S POSSIBLE.™

- What is LC/IMS/MS ?
  - Liquid Chromatography (LC)
  - Ion Mobility Separation (IMS)
  - Mass Spectrometry (MS)
- Combined analytical technique with unparalleled separation and identification
- Applications
  - Biopharmaceutical
  - Life Sciences
  - Food safety
  - Environmental Protection
  - Clinical
  - Chemical Materials

Overview of LC/IMS/MS

- LC/IMS/MS instruments produce and measure ions of the analytical sample
- Output is a 4D plot of ion intensity vs time, mass to charge ratio (m/z), and ion mobility (drift)



Overview of LC/IMS/MS

Waters
THE SCIENCE OF WHAT'S POSSIBLE.™

- How the data is acquired ?
  - LC/IMS/MS instruments generate mass scans of m/z values (~5 ms)
  - 200 consecutive scans represent 200 mobility (drift) values at a point in time
  - Acquire blocks of 200 scans at each time value
- Scan data show sharp peaks, each corresponding to an ion
- Data is very noisy
- Scans have many zero intensity values (sparse)
- Scans are stored in compressed form
  - Simple compression (no zeros)
  - Similar to Yale sparse matrix compression

The Data Size Problem

Waters
THE SCIENCE OF WHAT'S POSSIBLE.™

- Uncompressed data size is huge
  - Typical analysis: 2,000 time values x 200,000 m/z values x 200 drift values
  - 80 billion data points
- This is often larger and will increase in the near future
- Even when compressed, data size is large (several GB)
  - Can’t store entire data in device memory (single GPU)
  - May even be a problem to store entire data in main memory
- Data must be retrieved and processed in reasonably sized tiles
- But there are minimum tile dimensions due to nature of computation
  - Tiles can’t be made arbitrarily small
  - Challenge for GPU processing

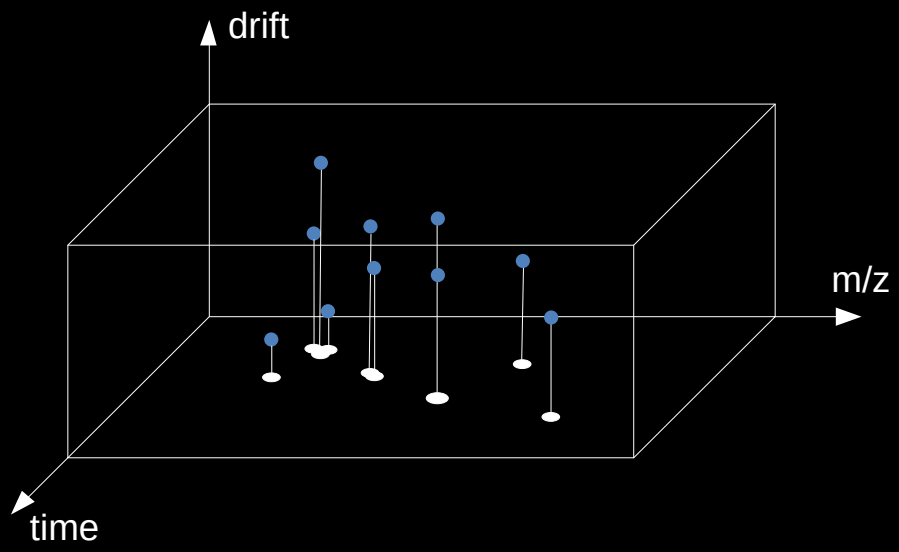
Objective of LC/IMS/MS Data Processing

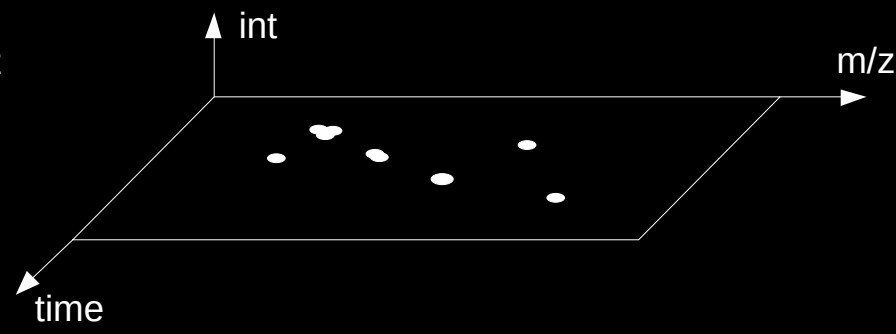
Waters
THE SCIENCE OF WHAT'S POSSIBLE.™

- Ion Detection
  - Find the location and intensity of every ion (every peak) in the volume of output data
  - Compute properties of each peak found
  - Generate a peak list
- Given the data is noisy, it has to be filtered to find peaks precisely
  - Run convolution filters in all three axis (time, m/z, and drift)
- Computing convolution filters in the entire (uncompressed) data is impractical
  - Too many data points
  - Need alternate method

Ion Detection in the GPU

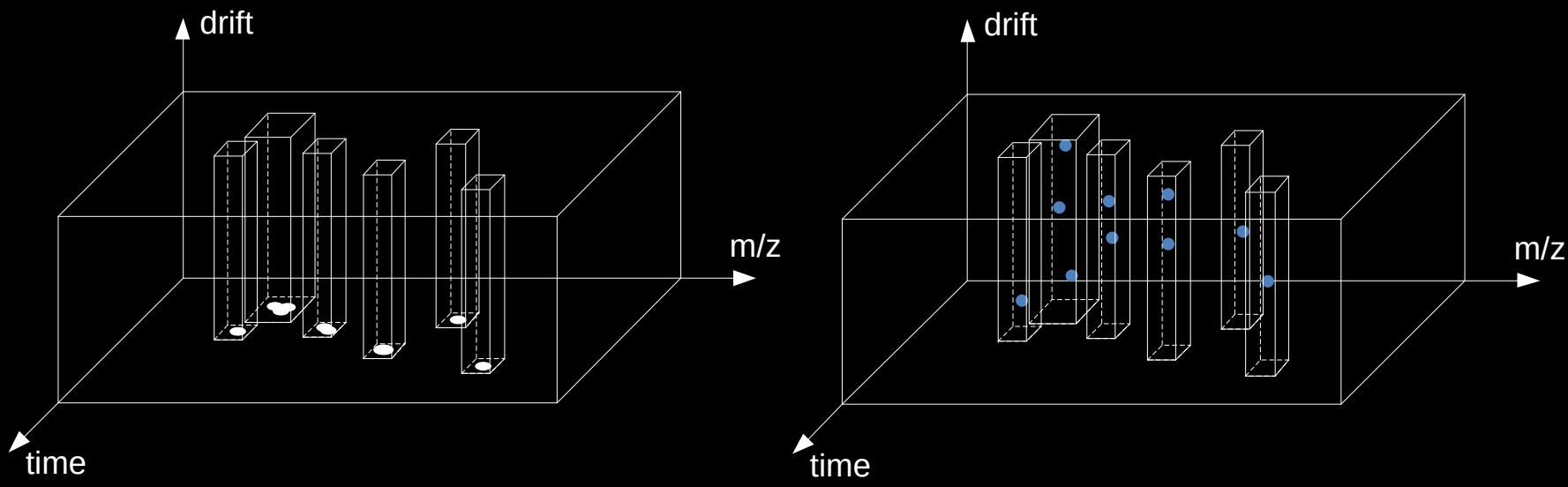
- Broken down into two major steps
  - 3D and 4D Processing
  - Takes advantage of sparsity and nature of the data (how mobility is acquired)
- 3D Processing
  - 4D data is collapsed to 3D by summing data along the drift axis
  - Data still is sparse
  - Peaks are detected in 3D space





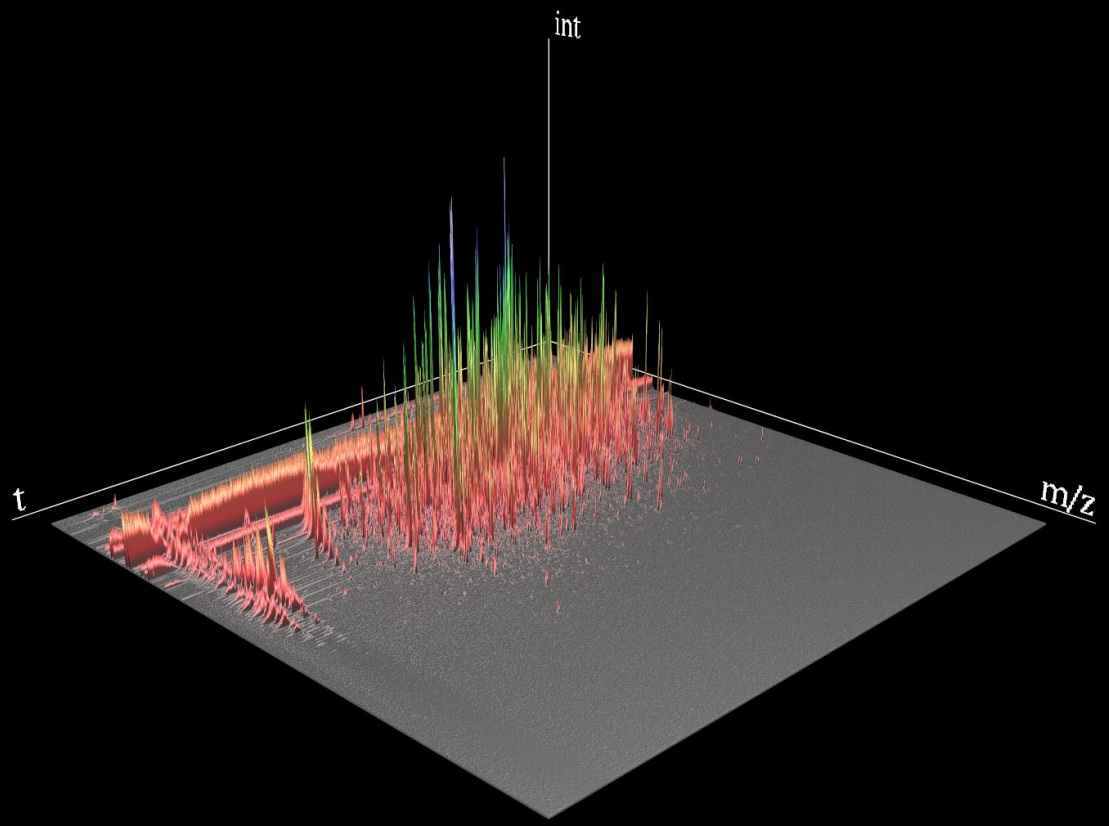
Ion Detection in the GPU

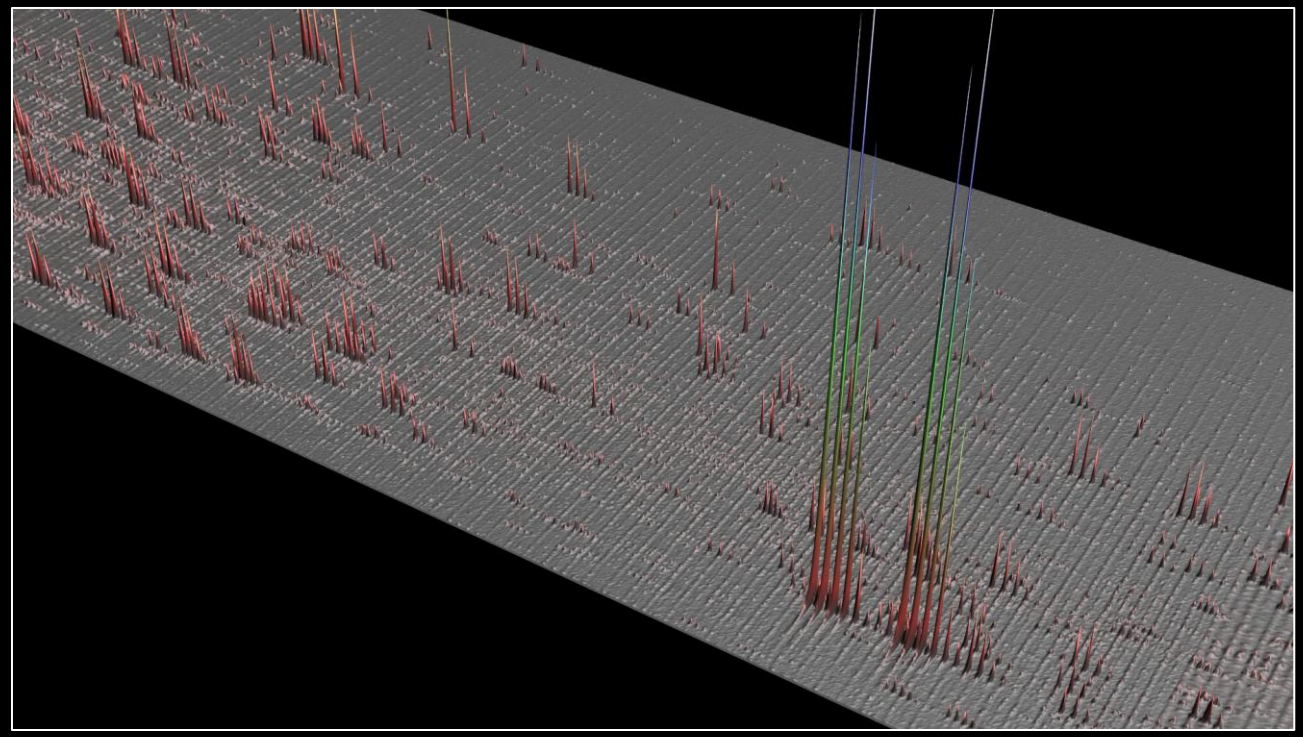
- 4D Processing
  - Uses peaks detected in 3D Processing
  - Builds a volume of 4D data around each peak
  - Peaks are detected in 4D space inside each volume



3D Data Visualization

- Data size: 3745 (time) x 172289 (mass) x 200 (drift) = 129 billion data points
  - Compressed data file: 9 GB
  - Data points in 3D: 645 million





3D Data Processing

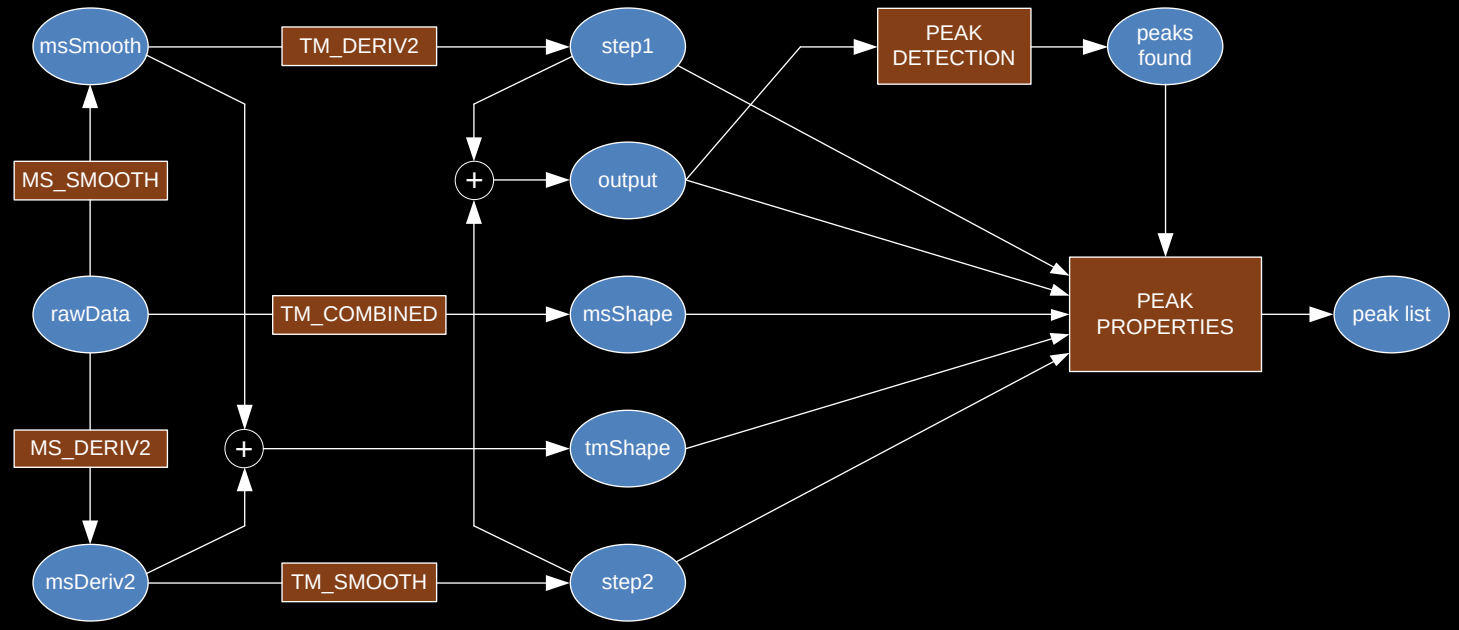
3D Processing Steps

- Pre-processing
  - Computes filter coefficients and other required parameters
  - One time computation done in the CPU
- Read raw data
  - Decompresses the data and collapses (sum) the drift axis
  - Currently done in the CPU
- Filter data
  - Runs convolution filters in the mass and time axes
- Peak detection
  - Finds peak apices in the filtered data
- Peak properties computation
  - Computes properties of each detected peak

Data Flow

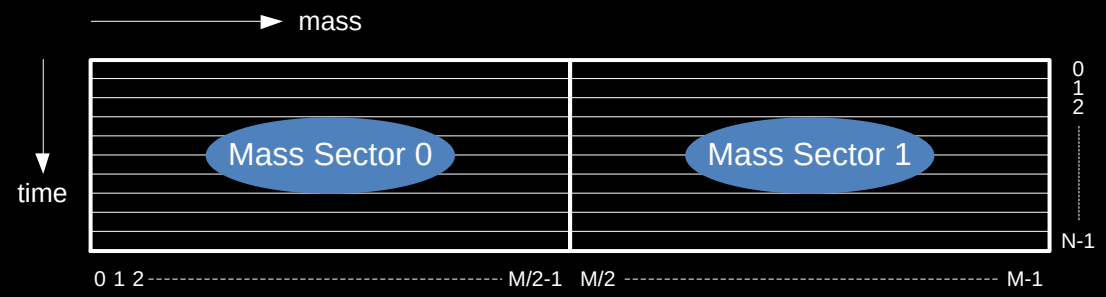
- 3D Processing Filter
  - Eight types of data
  - Five convolution filters of three types
    - Smooth, second derivative (deriv2), and combined (smooth+deriv2)
  - Filter along both axes
    - mass (ms) and time (tm)

- 3D Processing Peak Detection and Peak Properties
  - Use five of those eight types of data
  - Generate the peak list



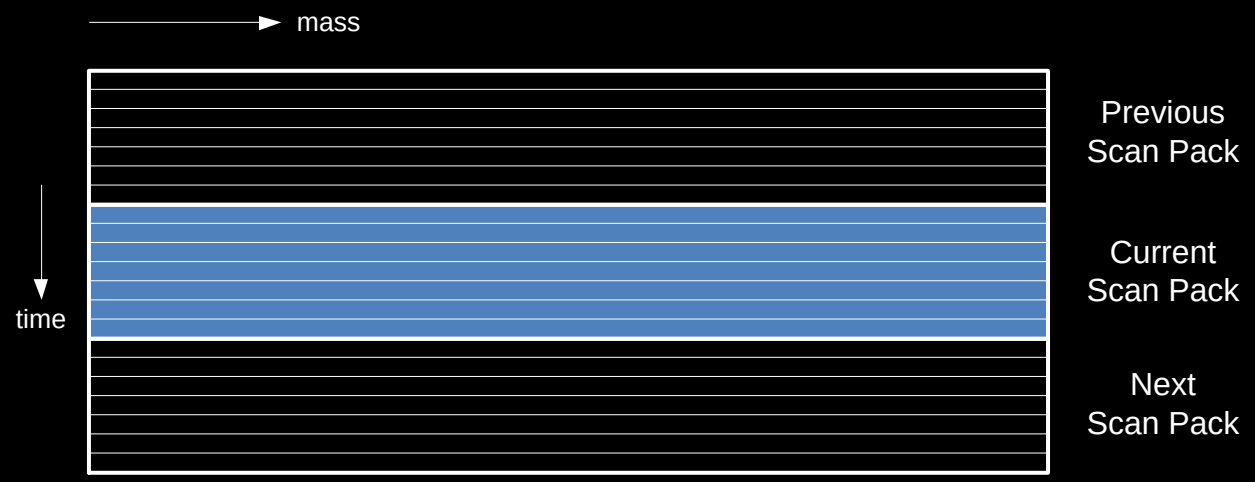
Scan Packs

- Data is grouped in Scan Packs
  - Consecutive mass scans
  - Covers a section of the time axis
  - Number of scans depend on data to process (64 to 100 typical)
- If needed, each scan pack is broken into Mass Sectors
  - Covers a section of the mass axis
  - Number of sectors depend on device memory available (1 or 2 sectors typical)



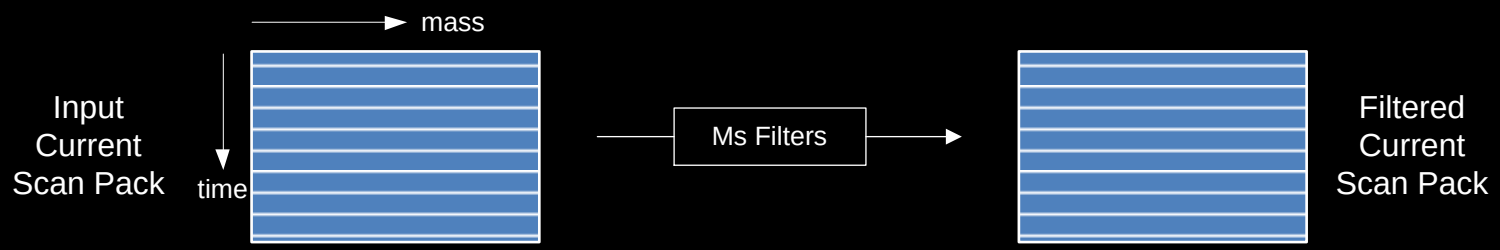
Scan Packs

- There are scan packs of the eight types of data mentioned
  - (raw data scan pack, msSmooth scan pack, …)
- Processing is done one scan pack at a time
  - One input scan pack, one output scan pack
- At least two consecutive scan packs are maintained in device memory for each type of data (three for some)



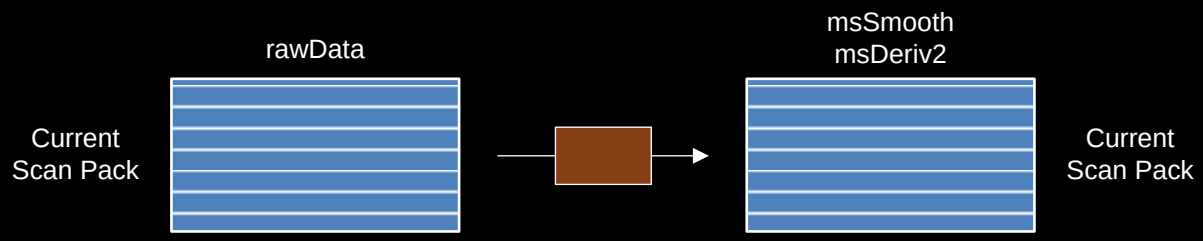
Scan Packs

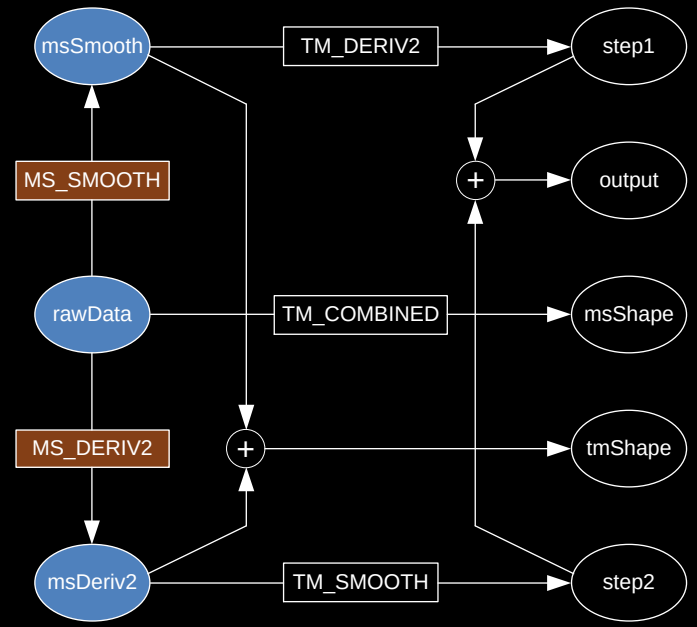
- For each output data point, convolution filters require as many input data points as the number of filter coefficients
  - The filter boundary is defined here as the number of coefficients minus one
  - Need additional "boundary" input data points after the point being filtered
- For filters in the mass direction
  - Filter boundary only impose some overlap of mass sectors
  - But scan packs can be filtered independently



Scan Packs

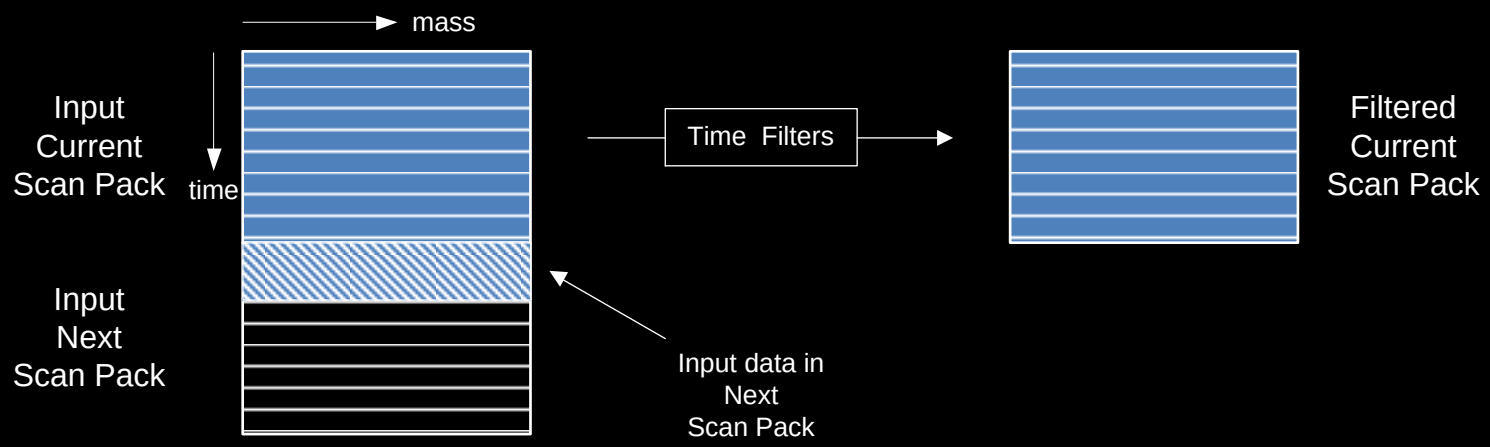
- This applies to the two mass axis filters
  - Only the current scan pack must be in memory





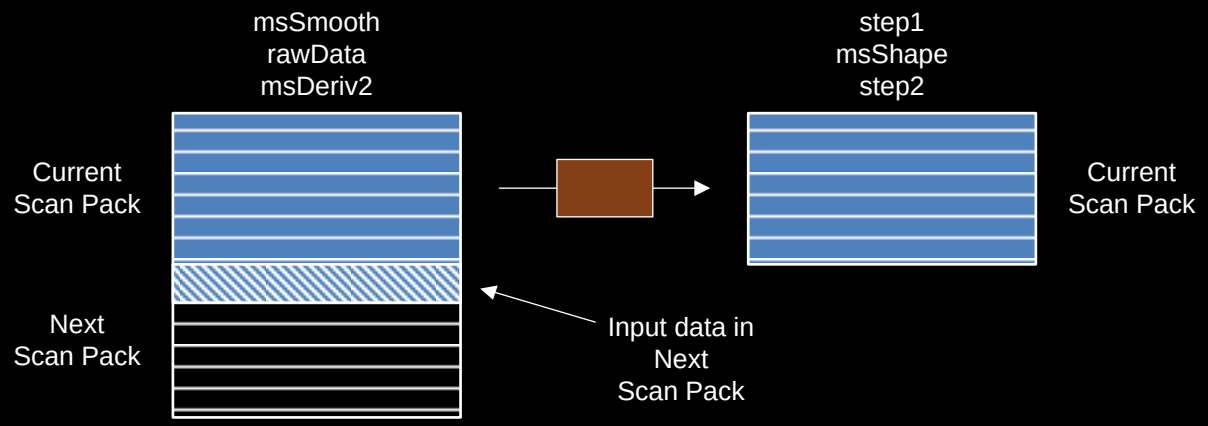
Scan Packs

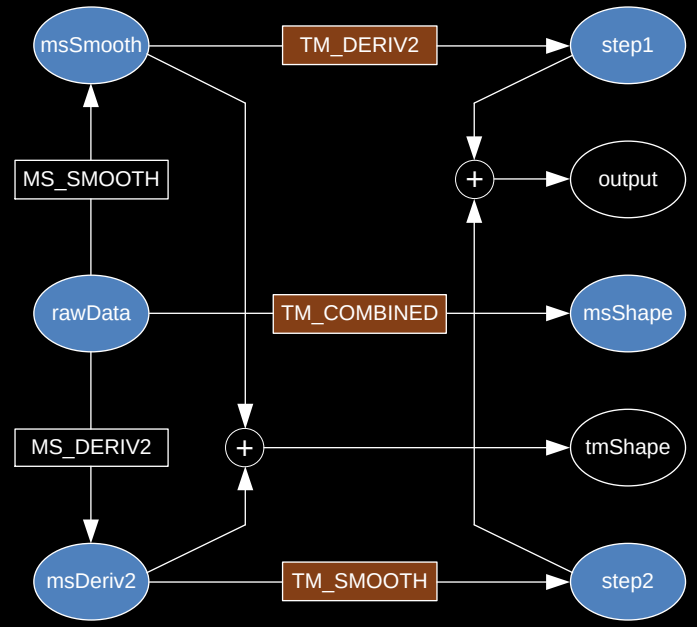
- For filters in the time direction
  - Filter boundary requires part of the next scan pack to filter the current scan pack
  - Scan packs cannot be filtered independently



Scan Packs

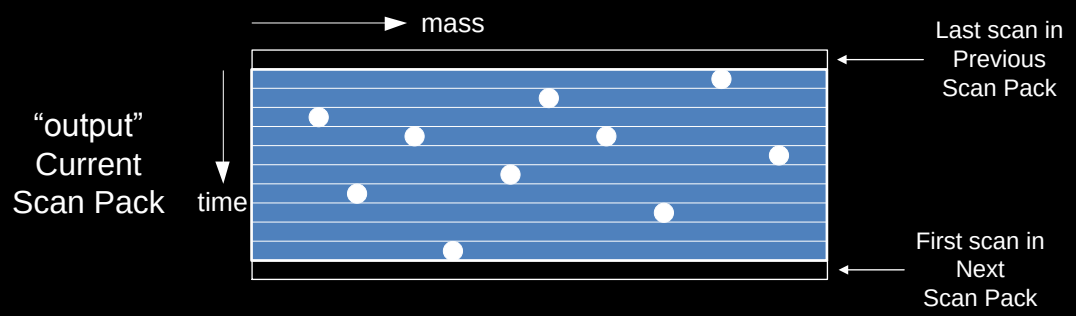
- This applies to the three time axis filters
  - Both the current and the next scan packs must be in memory





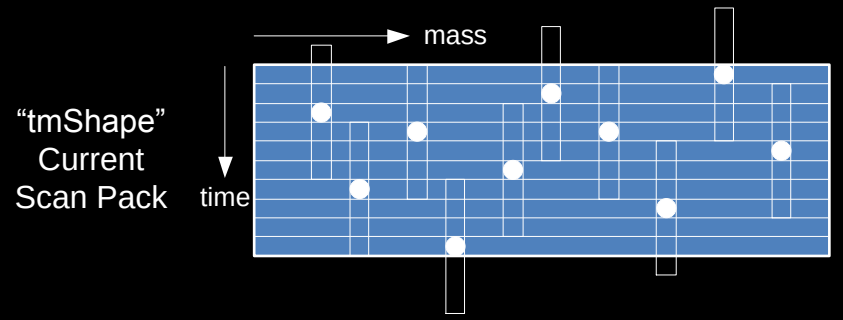
Scan Packs

- Peak detection uses three consecutive scans of the “output” scan pack
  - We need data in the previous and next scan packs to find peaks at the edges
  - At least one scan in these scan packs of “output” data must be in memory



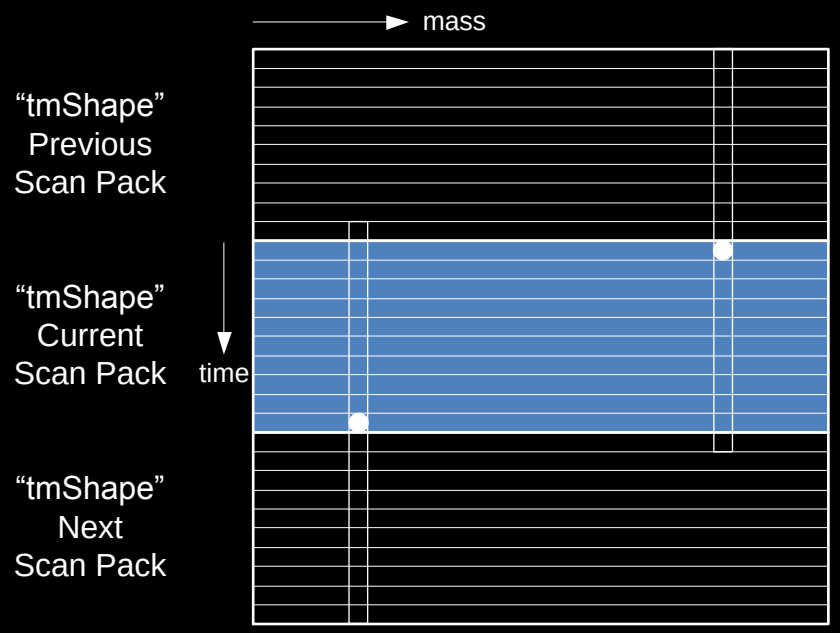
Scan Packs

- Similarly, peak properties computation uses the “tmShape” scan pack to extract a profile or “shape” of each peak along the time axis
  - We need data in the previous and next scan packs to extract the profile of peaks found near the edges
  - The previous, current, and next scan packs of “tmShape” data must be in memory



Scan Packs

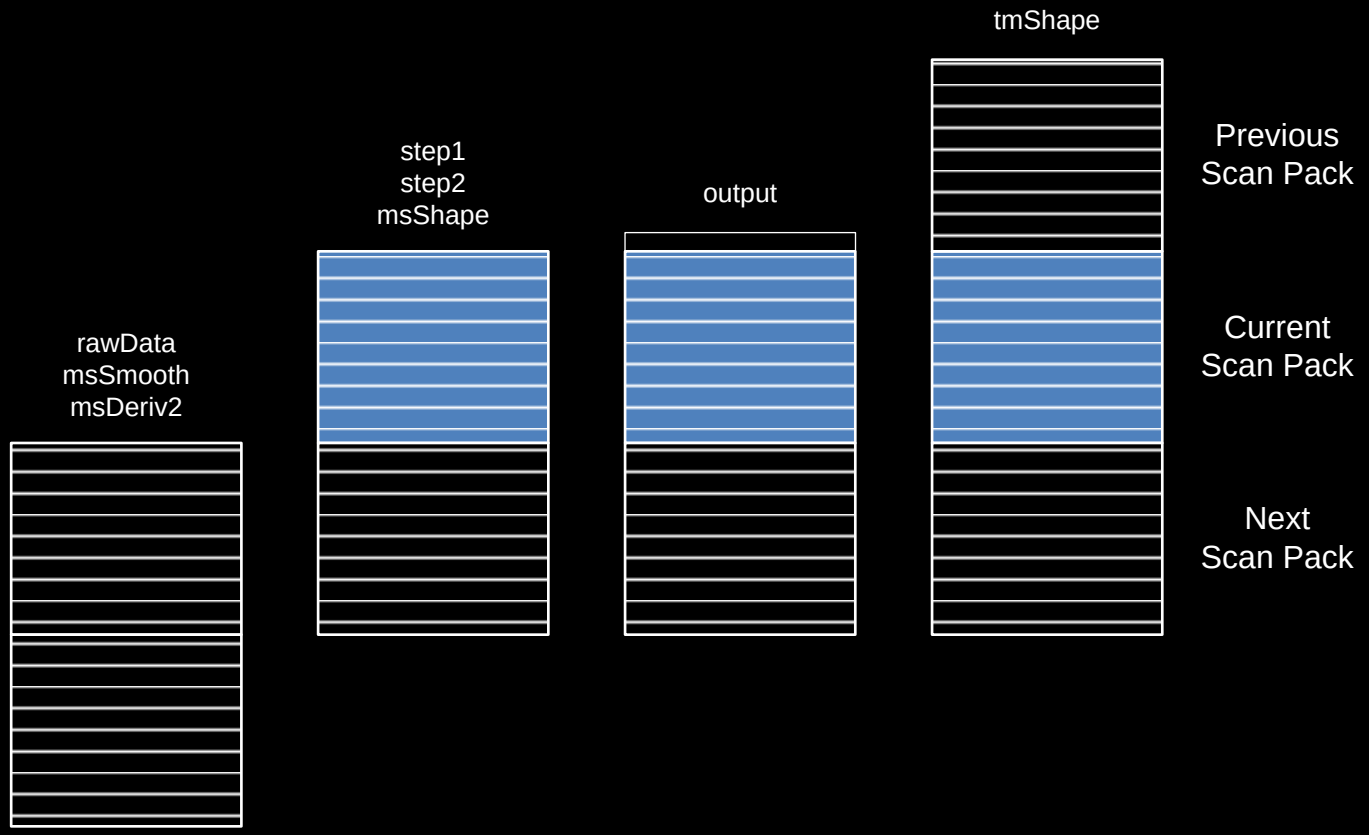
- The length of this profile determines the minimum scan pack size
  - Guarantees we don't need more than three "tmShape" scan packs



Time shape profile length: 21
Minimum scan pack size: 10

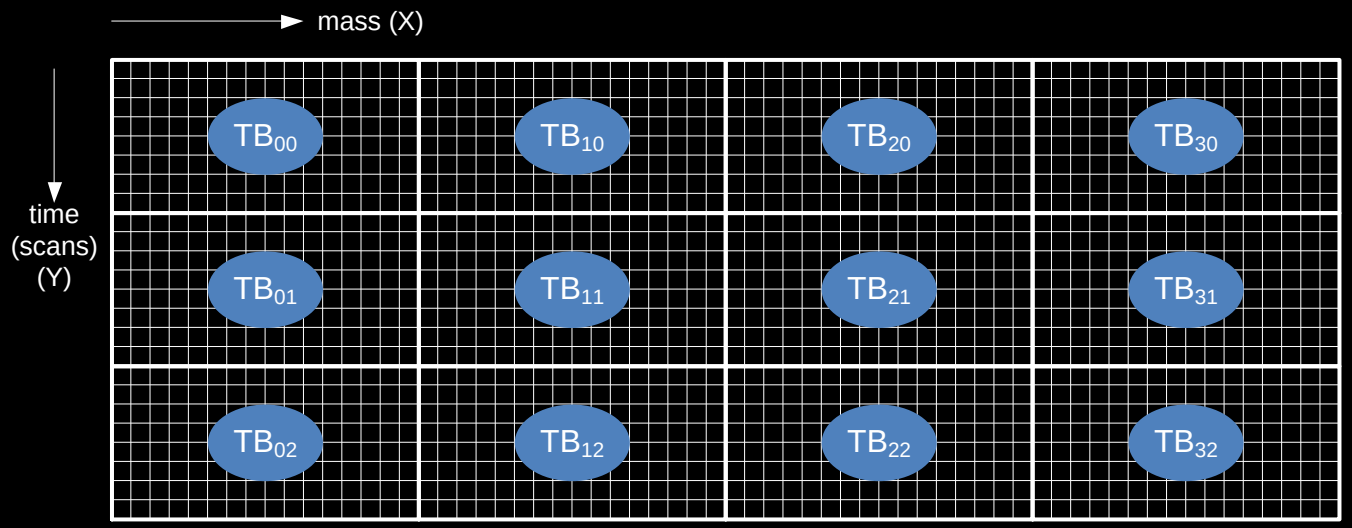
Scan Packs

- Resultant device memory needed for the eight types of data
- All scan packs have to be aligned
  - Peak properties computation need matching points for each data type
  - All scan packs are delayed to keep the "next" scan pack in memory



Data Filter

- Two-dimensional thread blocks are used
  - Mass along the x dimension
  - Time along the y dimension
  - Currently each thread filters one data point
- Prefer to exactly fit a number of thread blocks in the scan pack
  - No wasted threads
  - The scan pack size is set to meet this requirement (our choice)

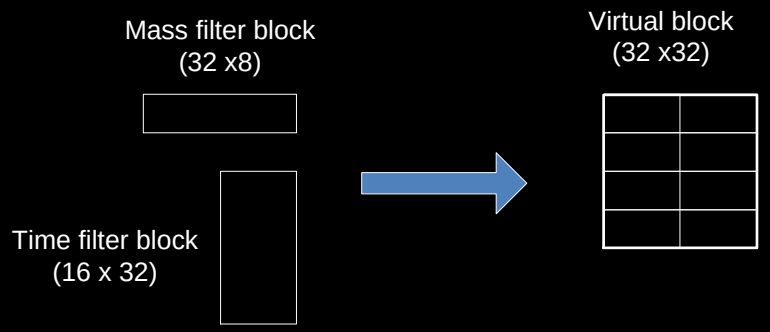


Data Filter

- Each thread uses almost the same input data points as adjacent threads in the direction of the filter
  - Nature of convolution filters
- Dimension the thread blocks to maximize data reuse
  - blockDim.x > blockDim.y for mass filters (along mass axis)
  - blockDim.y > blockDim.x for time filters (along time axis)

Data Filter

- Thread blocks dimensions are chosen
  - Dependent on the GPU used
  - To favor data reuse as described
- But we still prefer to exactly fit a number of blocks in the scan pack
  - Two types of blocks with different form factor
- Compute a virtual block with dimensions the least common multiple of both blocks
  - Use the virtual block only to
    - Set the scan pack size to be a multiple of the virtual block y dimension
    - Pad the scans with zeros to a multiple of the virtual block x dimension



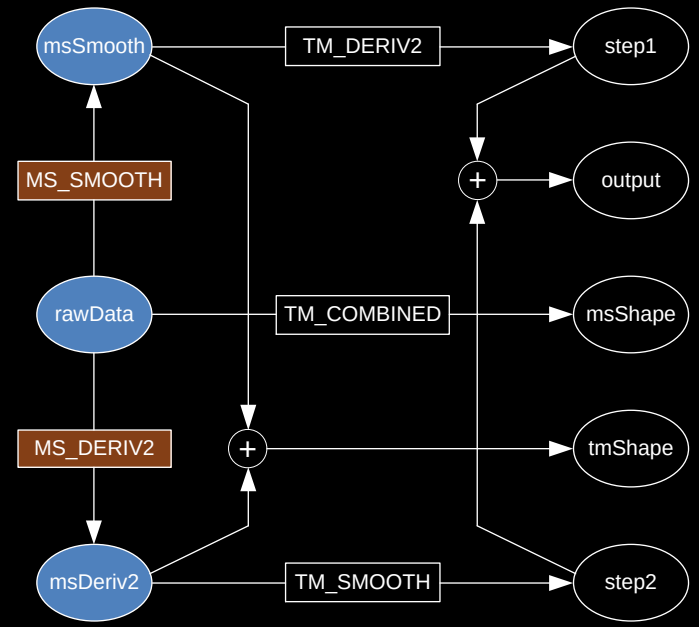
Mass Axis Filter

- The mass axis convolution filters (ms filters)
  - Use a different set of filter coefficients for each point in the mass axis
  - The number (odd) of coefficients increases with mass
  - All coefficient sets organized in a matrix in device memory
    - Coefficients are symmetrical (only half the coefficients stored)
  - Two matrices needed, one for each filter type (smooth and deriv2)

Mass point	Coefficients matrix				Num coeffs
0	$C0_0$	$C0_1$	0	0	3
1	$C1_0$	$C1_1$	0	0	3
2	$C2_0$	$C2_1$	0	0	3
3	$C3_0$	$C3_1$	0	0	3
4	$C4_0$	$C4_1$	$C4_2$	0	5
5	$C5_0$	$C5_1$	$C5_2$	0	5
6	$C6_0$	$C6_1$	$C6_2$	0	5
7	$C7_0$	$C7_1$	$C7_2$	0	5
8	$C8_0$	$C8_1$	$C8_2$	$C8_3$	7
9	$C9_0$	$C9_1$	$C9_2$	$C9_3$	7
10	$C10_0$	$C10_1$	$C10_2$	$C10_3$	7
11	$C11_0$	$C11_1$	$C11_2$	$C11_3$	7

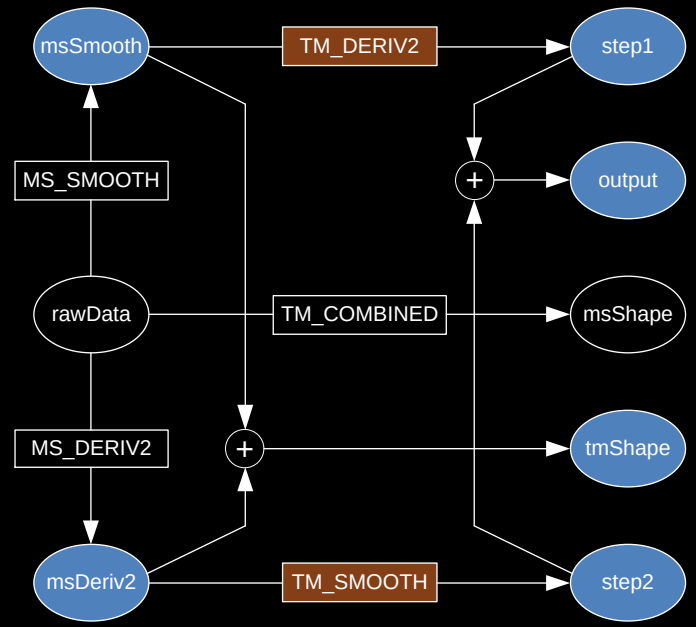
Mass Axis Filter

- One kernel computes both mass axis filters in one invocation
- Raw data is copied to shared memory
  - Shared memory input is used for both filters
- Both coefficient matrices are bound to linear textures
  - Too big for 2D textures



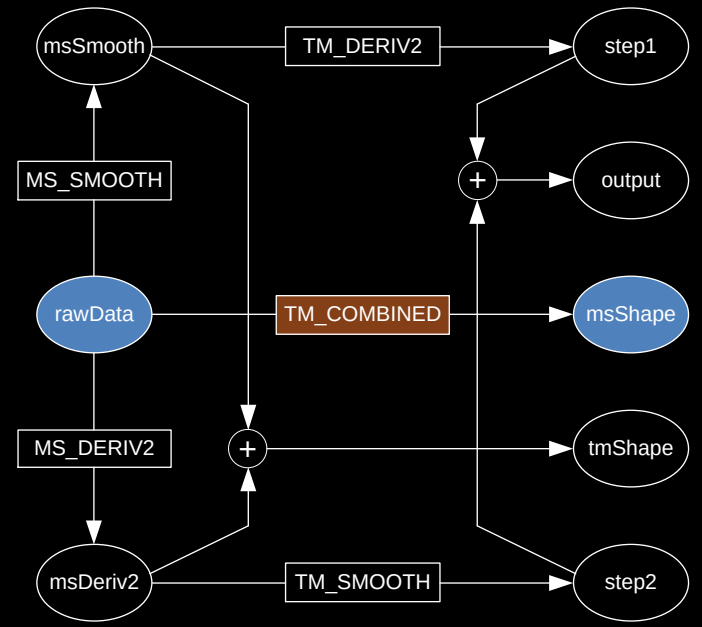
Time Axis Filter

- The time axis convolution filters
  - Use the same set of filter coefficients for all points in the time axis
  - Three coefficient sets needed, one for each type of filter
- One kernel computes two of the three time axis filters
  - Also computes the two averaged outputs to avoid additional reads



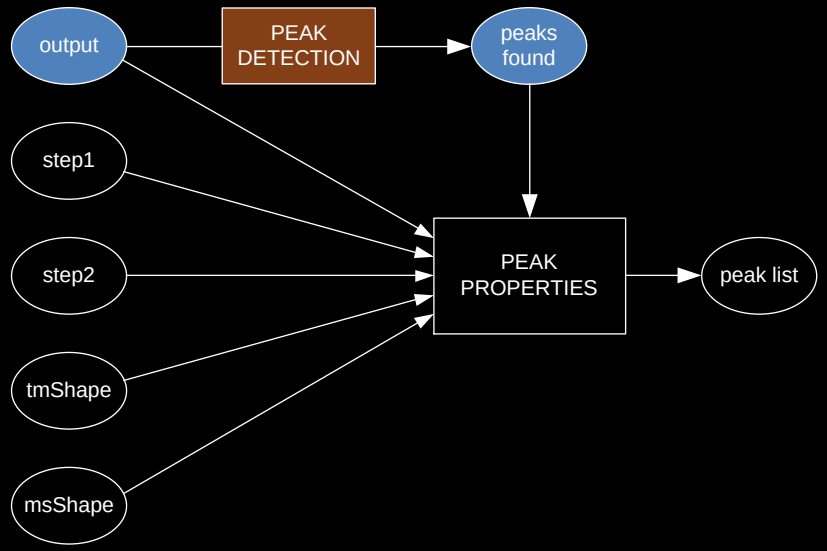
Time Axis Filter

- A second kernel computes the third time axis filter
- Each filter input data is copied to shared memory
- All filter coefficients are copied to constant memory



Peak Detection

- Finds peak apices in the “output” scan pack



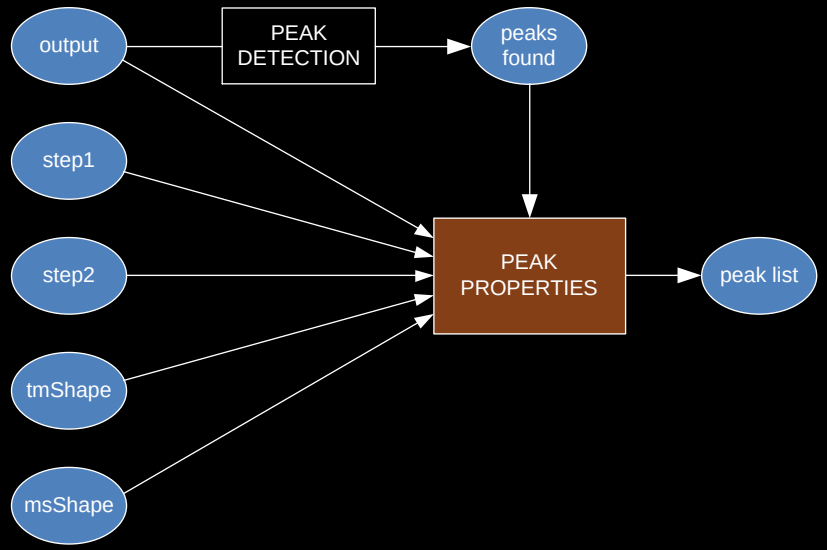
- Apex found at a point if it is a local maxima
  - Intensity is above a threshold
  - Intensity is greater than its eight neighbor points
- Each detected peak identified by its coordinates within the scan pack

Peak Detection

- Peak detection kernel
  - Use square thread blocks to maximize data reuse
  - “output” scan packs read from textures
  - Each thread tests one point
- Peak detection array
  - An array of detected peak coordinates
- A counter is incremented on each detected peak
  - Using an atomic operation
  - Determines the location of the peak in the peak detection array

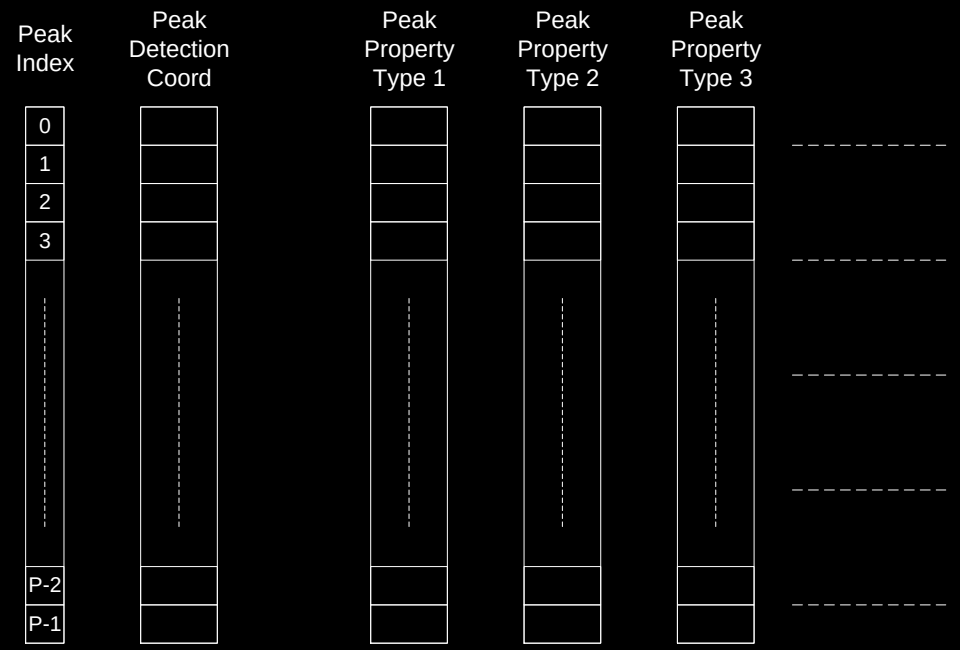
Peak Properties Computation

- Compute a set of properties of each detected peak
- Use the peak detection array and all five scan pack outputs from the filter step



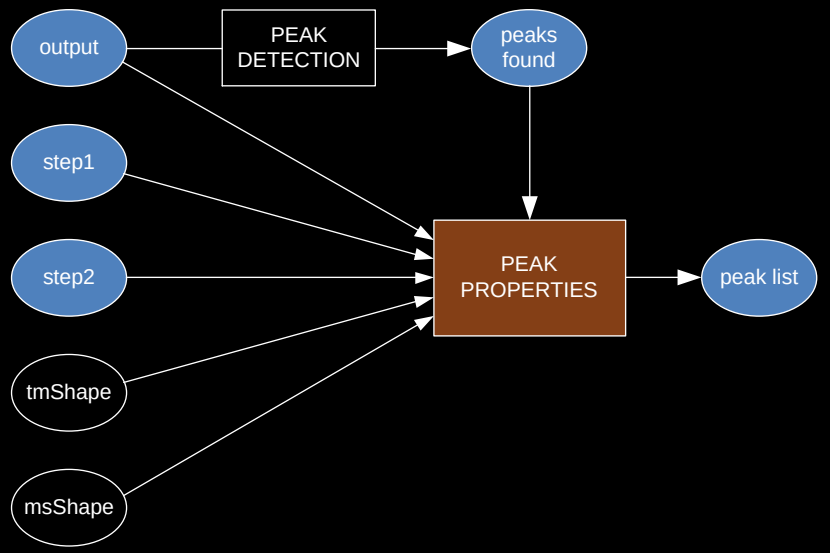
Peak Properties Computation

- Peak properties are saved in device memory arrays
  - One array per peak property type
  - All the same size (one element per detected peak)
  - Contain peak properties from detected peaks in all scan packs



Peak Properties Computation

- Two groups of peak properties
- A first group computed from “output”, “step1”, and “step2” scan packs
  - Intensity of the apex
  - Exact time and mass of the apex
  - Peak exclusion rules

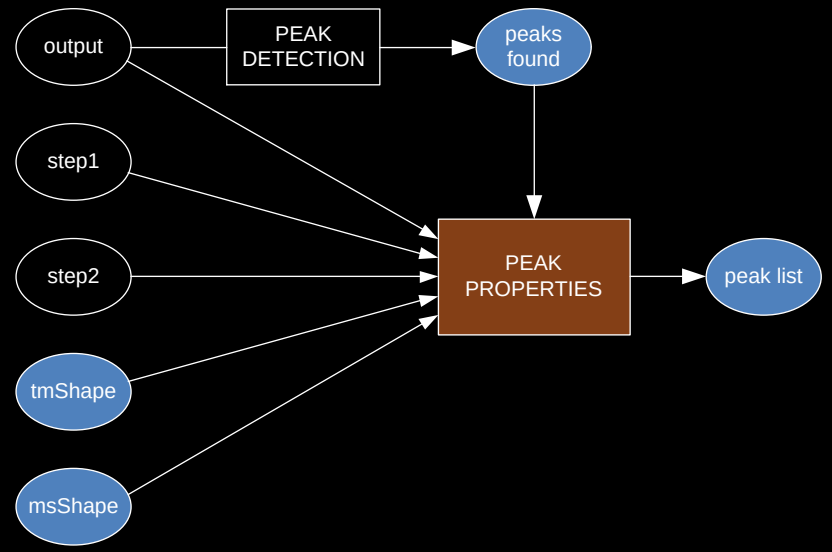


Peak Properties Computation

- One kernel computes the first group of peak properties
  - Use one-dimensional thread blocks and grid
  - Input scan packs bound to linear textures
  - Each thread computes properties for one peak
- The coordinates from the peak detection array
  - Used to read from the scan packs at the right location
- The index into the peak detection array
  - Used to address the peak property arrays

Peak Properties Computation

- A second group of peak properties computed from “tmShape” and “msShape” scan packs
  - Exact time and mass of apex (from profile)
  - Inflection points of peak
  - Start and stop points of peak
  - FWHM (full width at half maximum) of peak
  - Area of peak
  - Signal to noise ratio of peak
  - Additional peak exclusion rules

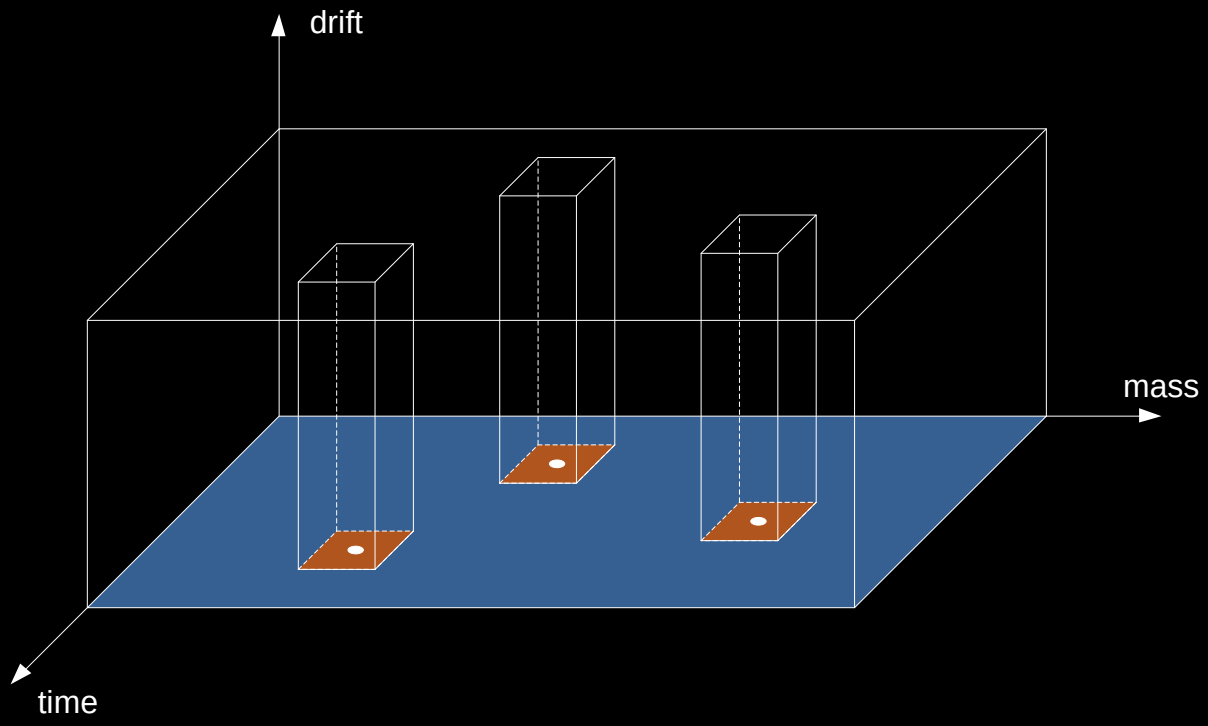


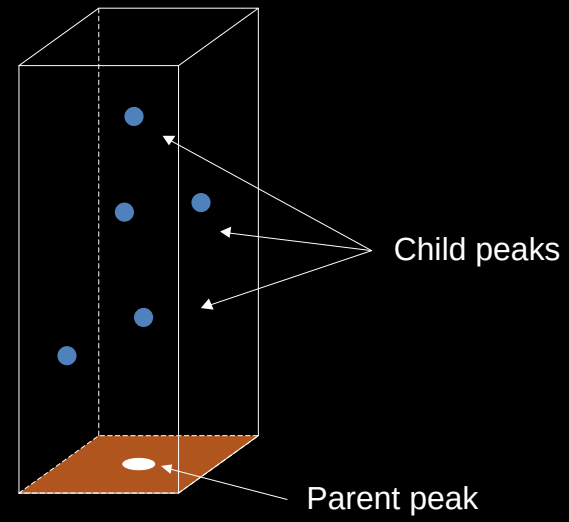
- These peak properties are computed twice
  - One set of properties in the time axis (using “tmShape”)
  - A second set of properties in the mass axis (using “msShape”)
- Computation of these peak properties is complex and will not be discussed here
  - Subject for a talk by itself
  - Several kernels involved

4D Data Processing

Parent Peaks

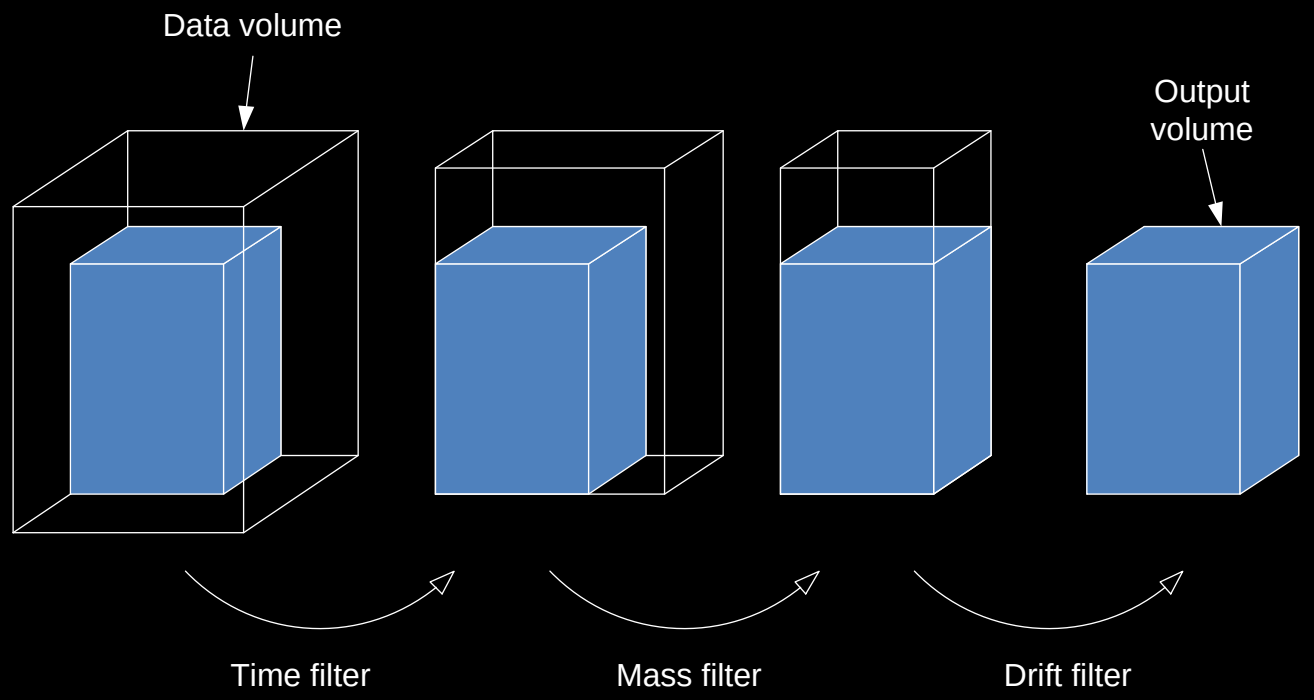
- Peaks detected in 3D Processing
- Determine where we search for peaks in 4D Processing (child peaks)
- For each parent peak, specify a search volume with dimensions
  - In mass and time set by the peak's start/stop points found in 3D processing
  - In drift is the entire drift range





Parent Peaks

- Parent peak volumes are filled with raw data, then filtered in each axis
  - Filter boundaries require to fill a larger volume (data volume)
  - Volume dimension in each axis reduced after the filter (output volume)



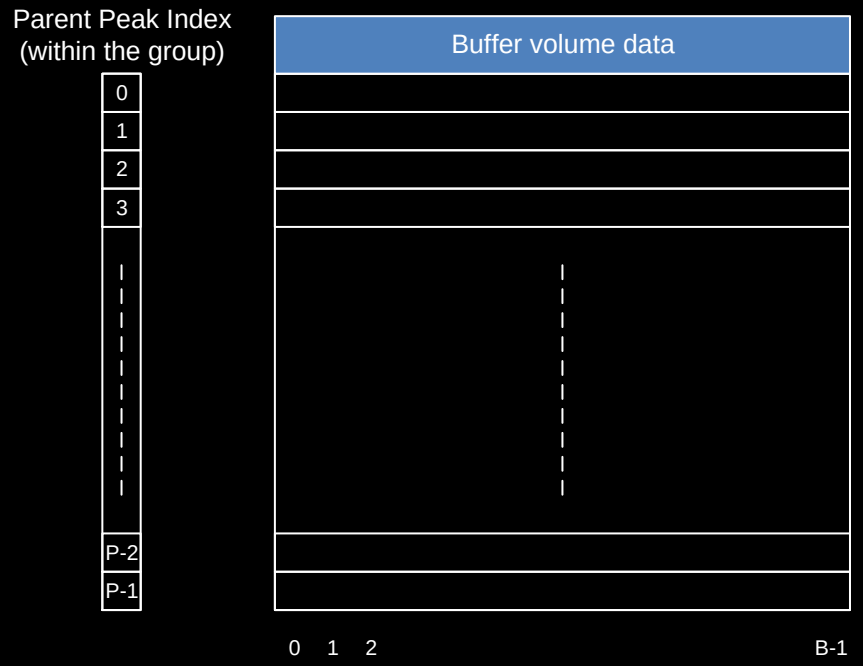
Parent Peaks

- A third type of volume (buffer volume) larger than any data volume
  - The same dimensions for all parent peaks
  - Common memory allocation size
  - Helps process volumes in parallel
- Each parent peak
  - Receives an allocation of “buffer volume” dimensions
  - Fills with raw data only its “data volume” dimensions
  - Filters the volume resulting in its “output volume” dimensions

Parent Peaks

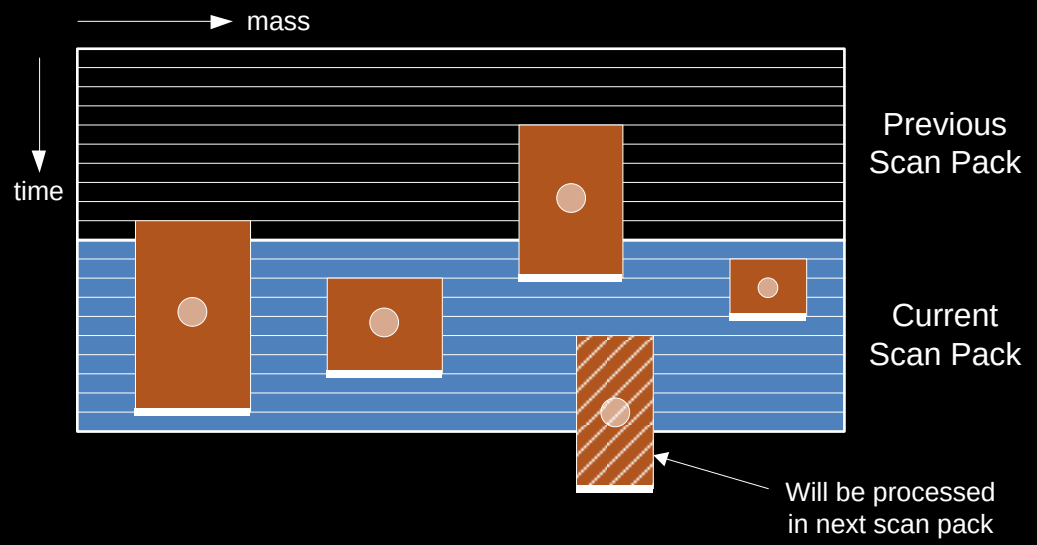
- Pack a group of "buffer volumes" in a device memory buffer
  - Help process a group of parent peaks in parallel
  - Three of these buffers required for processing
  - Large device memory allocation (keep the group small)

- Buffer volumes data organized in rows in the memory buffer
  - One row per parent peak
  - Kernels must convert between 3D and linear coordinates



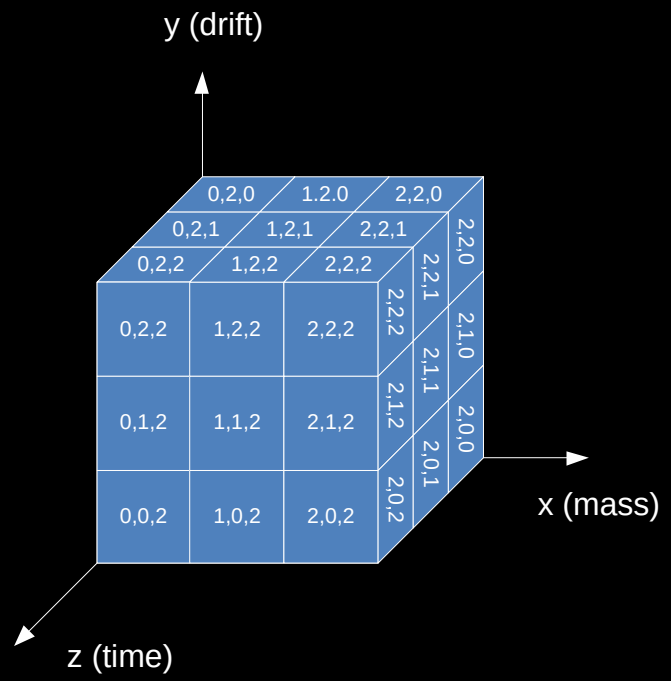
Data Organization

- Processing done in scan packs of compressed 4D data
  - Process all parent peaks with volumes falling inside the scan pack
  - Largest volume dimension in time determines scan pack size
  - Large device memory allocation (two scan packs required)
- Parent peaks sorted by stop time
  - Guarantees that all volumes with a stop time falling inside the current scan pack, can be processed with the current and the previous scan packs



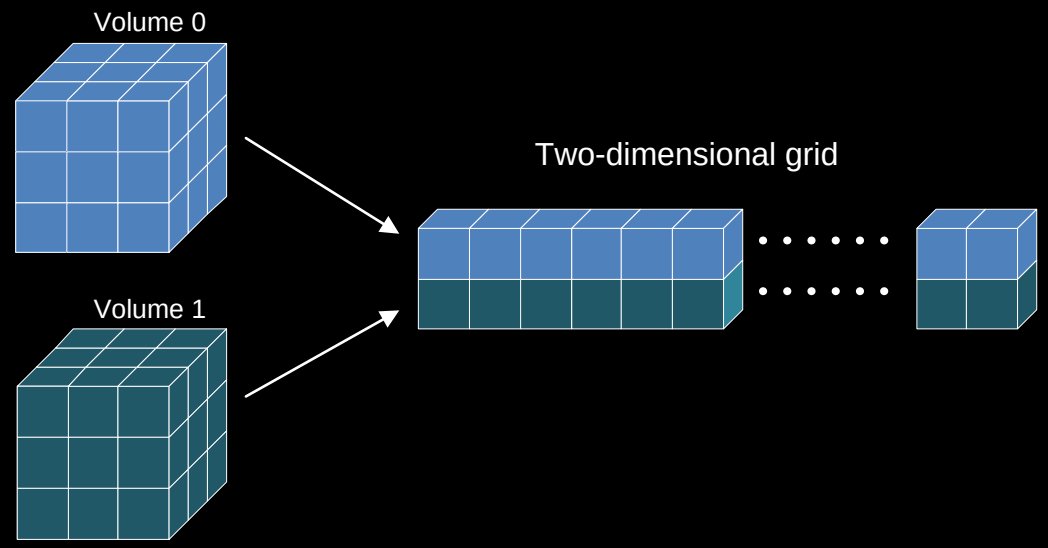
Thread Blocks

- Three-dimensional thread blocks and grid used to filter volumes
  - x (mass), y (drift), z (time)
  - Dimensions set to favor each type of computation
  - Each thread filters multiple data points
- Example of a volume using 27 thread blocks



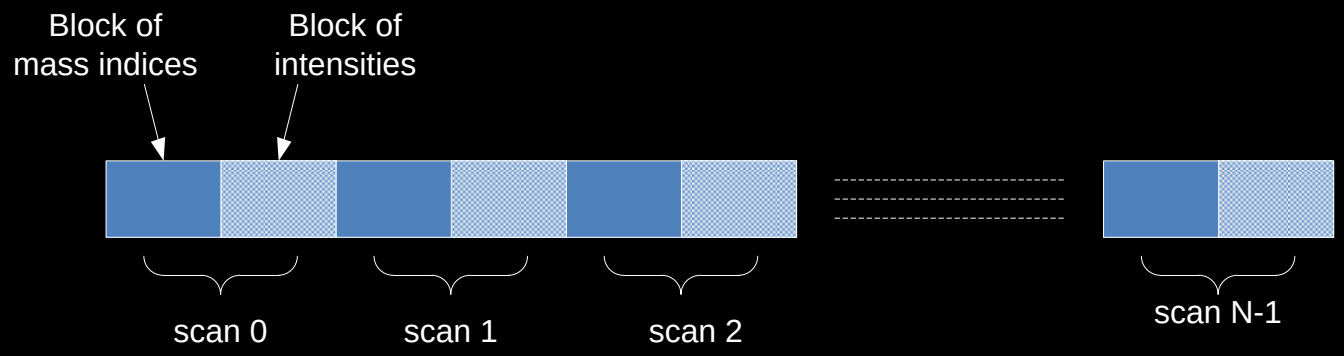
Thread Blocks

- However, this three-dimensional grid is virtual
  (design is prior to when three-dimensional grids were possible)
- The actual grid is two-dimensional and covers all volumes in a group of parent peaks
  - blockIdx.y = Parent peak index in the group
  - blockIdx.x = The 3D blocks of each volume distributed linearly
  - Kernels must convert from linear to 3D block indices



Fill Volumes

- Parent peak volumes are filled with raw data
  - From compressed scan packs
  - All parent peak volumes in a group (a volume buffer)
  - Decompression done in the GPU (only volumes, not entire scans)
- Raw data comes compressed without zero intensity points
  - All non-zero intensities from 200 mobility scans are packed in a block
  - A companion block of mass indices specify the location of the non-zero values
  - Each pair of these blocks make a regular mass scan (a point in time)



Fill Volumes

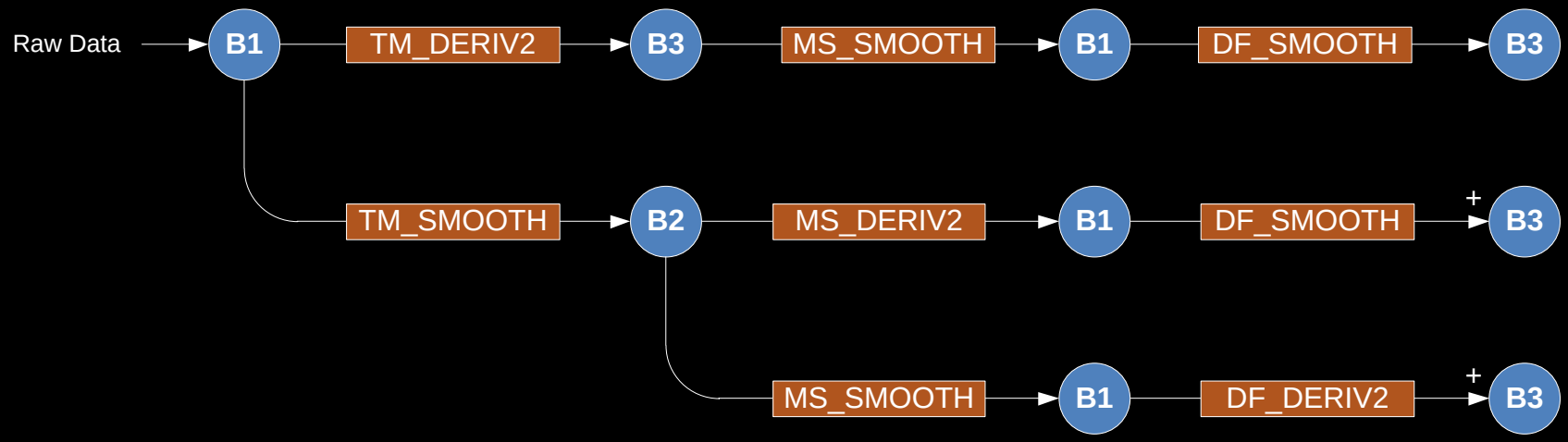
- All blocks from scans needed to build a scan pack are organized differently in device memory
  - All intensities blocks consecutive in an array
  - All mass indices blocks consecutive in other array
  - Easier to address mobility scans inside the kernel

Fill Volumes

- The kernel uses two-dimensional thread blocks and a one-dimensional grid
  - x (time), y (drift)
  - One block per parent peak
  - Each thread handles multiple points in the volume
- For each volume
  - Find the volume's start mass index in each mobility scan intersecting the volume
  - Use a binary search in the array of mass indices blocks
- Then, for each intersecting scan
  - Copy all non-zero intensities to the volume, at the positions determined by the mass indices
  - Stop the copy when the volume's stop mass index is encountered
  - Volume intensity set to zero in all mass indices not found

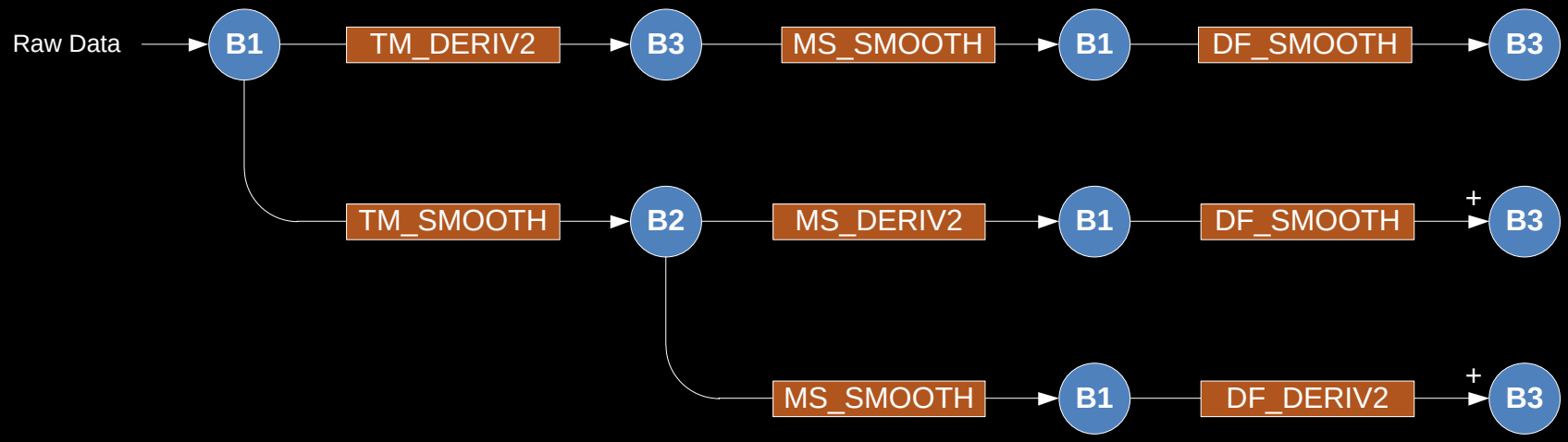
Filter Volumes

- All parent peak volumes in a volume buffer are filtered
  - Prepare the volumes for peak detection
  - Requires additional volume buffers for temporary results (reused)
- A complete filter sequence includes
  - Eight convolution filters in the three axes (time, mass, drift) (tm, ms, df)
  - Two types of filters (smooth and second derivative)
  - Three volume buffers (B1, B2, B3)



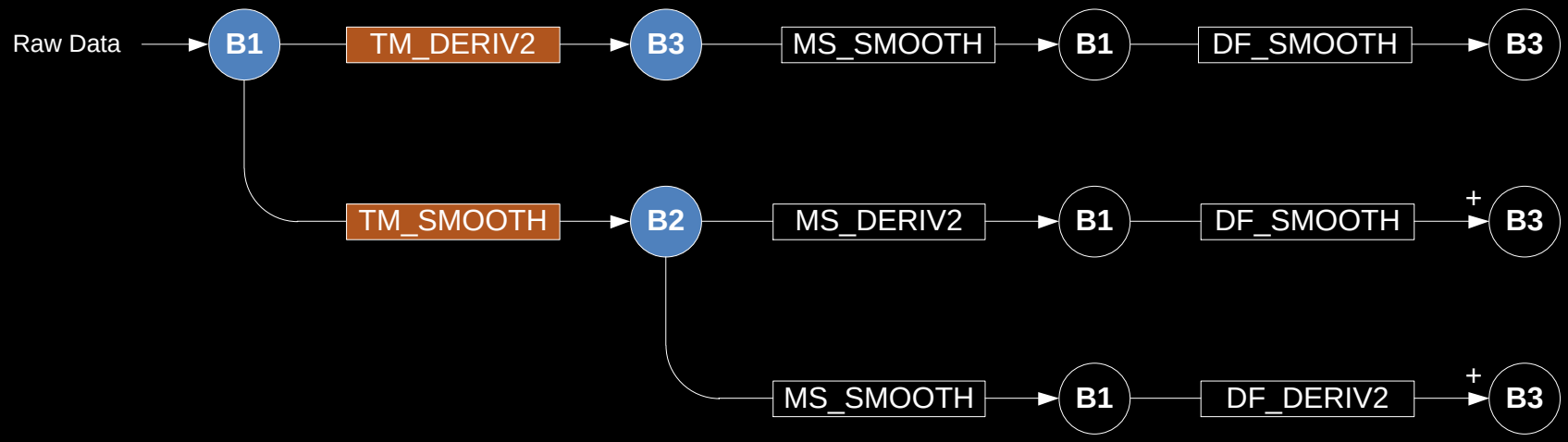
Filter Volumes

- Filter sequence
  - Three filter sub-sequences (three rows) that are run top to bottom
  - Each sub-sequence has a filter in each axis (from raw data to output)
  - The order of the filters is chrom → mass → drift in each sub-sequence
  - Each sub-sequence has only one second derivative filter
  - Final content of buffer B3 is the average of the three sub-sequences
    - B3 is written in the first sub-sequence, and accumulated in the other two
  - The minimum number of buffers required is three



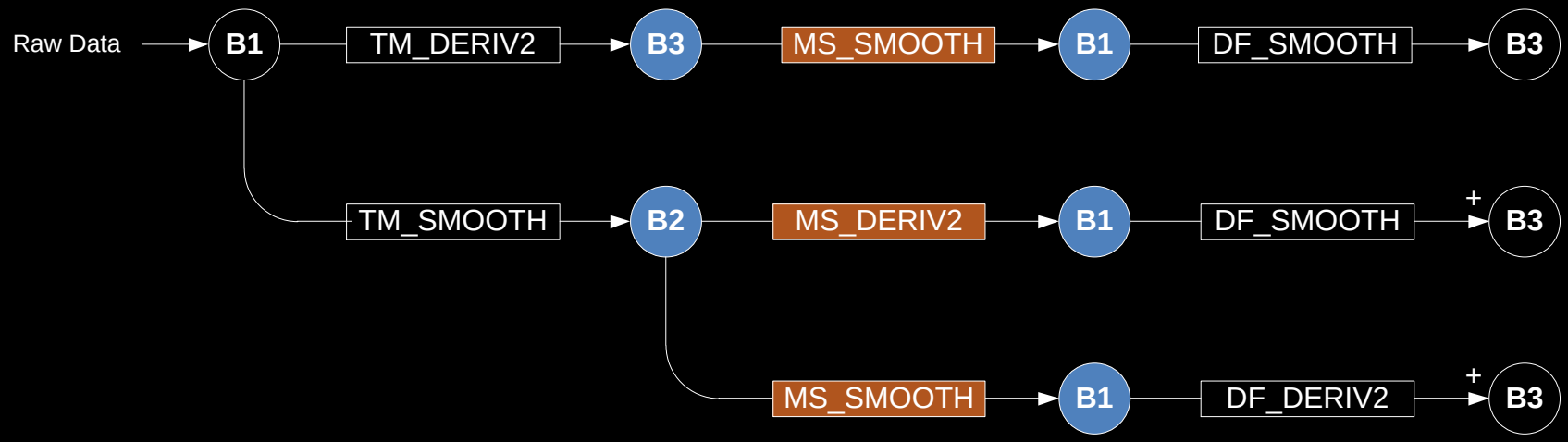
Filter Volumes

- Time filter kernel
  - Uses three-dimensional thread blocks and grid as described earlier
  - Launched only once per group of parent peaks
  - Computes both time filters from the raw data volume buffer
    - One input two outputs
    - Frees buffer B1 for reuse in next filter steps
  - Filter coefficients are the same as in 3D processing (constant memory)



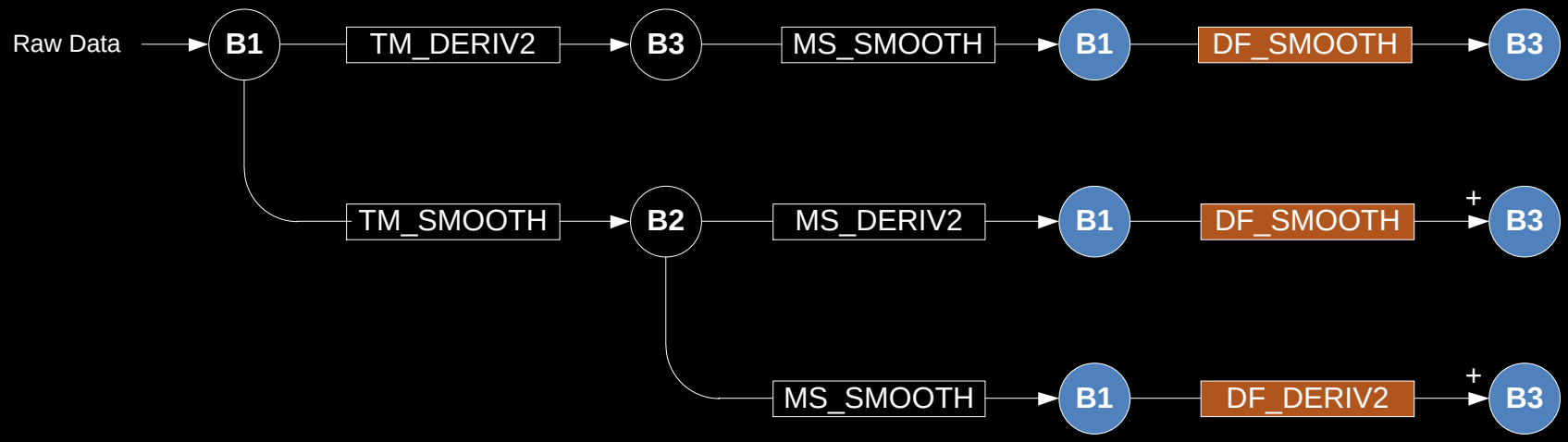
Filter Volumes

- Mass filter kernel
  - Uses three-dimensional thread blocks and grid as described earlier
  - Launched three times per group of parent peaks
  - Computes the smooth or the second derivative filter depending on the coefficients passed (one input one output)
  - Filter coefficients are the same as in 3D processing (a matrix in device memory)



Filter Volumes

- Drift filter kernel
  - Uses three-dimensional thread blocks and grid as described earlier
  - Launched three times per group of parent peaks
  - Computes the smooth or the second derivative filter depending on the coefficients passed (one input one output)
  - Filter coefficients are different for each drift value (a matrix in device memory)
  - The first launch writes to buffer B3 and the other two accumulate on B3



Peak Detection

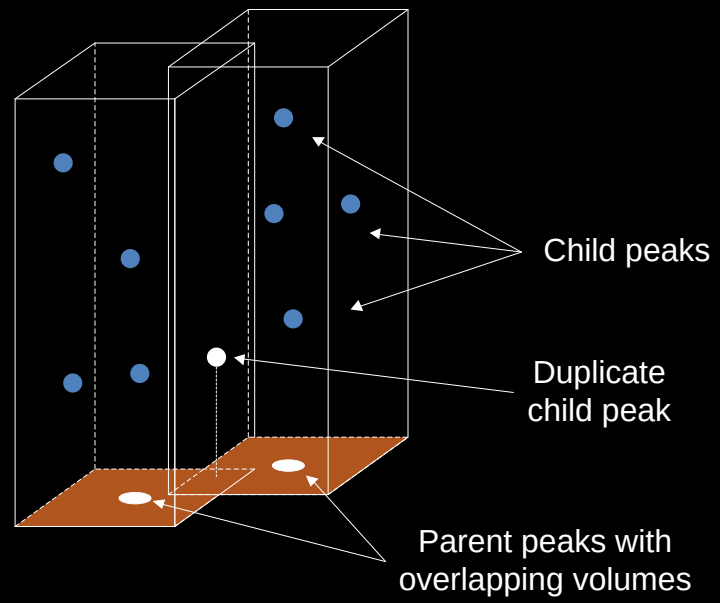
- Finds child peaks in filtered parent peak volumes
  - All volumes in buffer B3
- Apex found at a point if it is a local maxima
  - Intensity is above a threshold
  - Intensity is greater than its 26 neighbor points
- Each detected peak identified by its global coordinates within the data
- Peak detection array
  - A list of detected peaks coordinates
- Other arrays created
  - Intensity value of lower neighbor in each axis
  - Intensity value of higher neighbor in each axis
  - Distance from parent to child

Peak Detection

- Peak detection kernel
  - Uses three-dimensional thread blocks and grid as described earlier
  - Launched only once per group of parent peaks
- A counter is incremented on each detected peak
  - Using an atomic operation
  - Determines the location of the peak in the peak detection array

Duplicate Removal

- Remove child peaks detected in more than one volume
  - Parent peak volumes overlap
  - Duplicate child peaks have identical coordinates
  - Keep the one closer to its parent peak
  - Thrust used for this computation (sort and unique)



Peak Properties Computation

Waters
THE SCIENCE OF WHAT'S POSSIBLE.™

- Similar to peak properties computation in 3D Processing
- A set of properties of each detected child peak in each scan pack
  - Intensity and exact time, mass, and drift of the apex
- Peak properties are saved in device memory arrays
  - One array per peak property type
  - Arrays may be saved in host memory if number of child peaks is excessive
  - Millions of child peaks are common
- One kernel computes the peak properties
  - Use one-dimensional thread blocks and grid
  - Each thread computes properties for one peak
  - Use the peak detection array and the additional arrays saved during peak detection

Results and Conclusions

Results

- Application support is limited to Tesla C2050, C2070 and C2075
  - Requirement for large device memory and TCC mode
- Speedup varies for each processing step
  - Use double precision only when needed
  - Filters in 4D processing dominate computation time
  - Overall speedup limited by data I/O
- Speedup also vary for different types of LC/IMS/MS data and data size
- Comparison results with Xeon X5650 2.67GHz CPU (single threaded) and ECC mode off in GPU

	Average Speedup
Data filters	40x
Peak detection/properties	15x
Overall	8x

Conclusions

- The GPU is enabling in LC/IMS/MS Proteomics applications
  - Unacceptable long processing times in the CPU for many users
- Also enables real-time processing during data acquisition
  - Embed the GPU inside or near the instrument
- The GPU will be more important in the future as LC/IMS/MS data size increases
- Future work
  - Port to the GPU some processing steps still done in the CPU
  - Performance optimizations
  - Add support for features required by new Mass Spectrometry instruments
- Acknowledgements
  - Marc Gorenstein, Dan Golick

Waters
THE SCIENCE OF WHAT'S POSSIBLE.™

Questions?

jose.de.corral@waters.com